Copyright © 2004, Software Engineering Research and Performance Engineering Services. All rights reserved.

Web Application Scalability: A Model-Based Approach

Lloyd G. Williams, Ph.D.
Software Engineering Research
264 Ridgeview Lane
Boulder, Colorado 80302
(303) 938-9847
boulderlgw@aol.com

Connie U. Smith, Ph.D.
Performance Engineering Services
PO Box 2640
Santa Fe, New Mexico, 87504-2640
(505) 988-3811
http://www.perfeng.com/

Scalability is one of the most important quality attributes of today's software systems. Yet, despite its importance, scalability in applications is poorly understood. This paper presents a model-based view of scalability in Web and other distributed applications that is aimed at removing some of the myth and mystery surrounding this important software quality. We review four models of scalability and show how they relate to Web and other distributed applications. The applicability of these models is demonstrated with a case study.

INTRODUCTION

Web and other distributed software systems are increasingly being deployed to support key aspects of businesses including sales, customer relationship management (CRM), and data processing. Examples include: online shopping, processing insurance claims, and processing financial trades. Many of these systems are deployed on the Web or on dedicated networks using Web technologies such as J2EE.

As these businesses grow, the systems that support their functions also need to grow to support more users, process more data, or both. As they grow, it is important to maintain their performance (responsiveness or throughput) [Smith and Williams 2002]. Poor performance in these applications often translates into substantial costs. Customers will often shop elsewhere rather than endure long waits. Slow responses in CRM applications mean that more customer-service representatives are needed. And, failure to process financial trades in a timely fashion can result in statutory penalties as well as lost customers.

These forces combine to make scalability one of the most important quality attributes of today's software systems. Yet, despite its importance, scalability in applications is poorly understood and there is no generally accepted definition of scalability. In this paper, we will use the following definition:

> Scalability is a measure of an application system's ability to—without modification—cost-effectively provide increased throughput, reduced response time and/or support more users when hardware resources are added.

Scalability is a *system* property. The software architecture is a key factor in determining scalability. For example, if the software architecture is not able to use additional resources to increase throughput the system will not be scalable. The choice of execution environment is also very important. As we will see, the same workload executed on two different platforms can exhibit significantly different scalability properties.

Scalability in Web and other distributed systems is a complex problem. Removing a bottleneck changes the dynamics of the system and a different bottleneck may emerge to impose new limitations on the scalability. Thus, it is necessary to reevaluate the behavior of the system after making changes.

This paper presents a model-based view of scalability in Web and other distributed applications. It is aimed at removing some of the myth and mystery surrounding this important software quality. It reviews previous work on scalability and shows how it relates to distributed Web applications. This paper also provides a simple demonstration that extrapolations of "near-linear" results are likely to overestimate scalability. Finally, it offers a reformulation of Gustafson's Law that is more appropriate for evaluating Web application scalability and demonstrates that this law is applicable to some Web applications.

The analyses described here are part of the PASA^{SM} approach to the performance assessment of software architectures [Williams and Smith 2002]. They are employed in situations where scalability is a concern

and where the required measurements can be obtained.

We begin with a motivating example followed by a review of scalability measures and models. A simple case study then illustrates how these models apply to Web applications.

MOTIVATION

Consider the following data (published on the Web) for horizontal scalability of a sample application developed using commercial Web server software.

Table 1: Horizontal Scaling Data

Number of Nodes	Transactions per Second
1	66.6
2	126.4
3	178.2
4	235.6

The report also contains the statement: "Multiple Web servers scale near-linearly (80–90%) at this level using [Product X]." While this statement is, at a casual glance, true it is also potentially dangerously misleading. A straightforward extrapolation to higher numbers of processors would give the scalability shown by the dashed line in Figure 1.

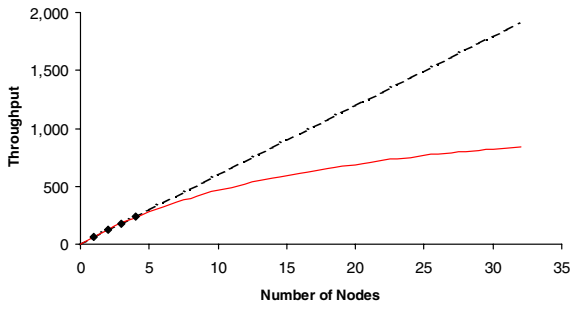

Figure 1: Extrapolated Scaling

This type of extrapolation is all too common in our experience. However, as we will see, the actual scalability of this system is more likely to follow the solid curve. The difference between these two predictions is significant. It could mean the difference between successful deployment of the application and financial disaster.

This example underscores the importance of obtaining a thorough understanding of the scalability characteristics of a system before committing resources to its expansion. One way to accomplish this is to determine—through analysis of measured data—whether the behavior of the system fits that of a known model of scalability. Once there is a degree of confidence in how well a model describes the system, extrapolations such as the one above become less risky.

SPEEDUP AND SCALEUP

Speedup is a measure of the reduction in time required to execute a fixed workload as a result of adding resources such as processors or disks. Speedup is the most common metric for evaluating parallel algorithms and architectures. *Scaleup*, on the other hand, is a measure of the increase in the amount of work that can be done in a fixed time as a result of adding resources. Scaleup is a more relevant metric for Web applications where the principal concern is whether we can process more transactions or support more users as we add resources.

Although they might appear to be very different metrics, speedup and scaleup are really two sides of the same coin. Clearly, if we can execute a transaction more quickly, we can execute more transactions in a given amount of time. More formally, the speedup is given by:

$$S(p) = \frac{T(1)}{T(p)}$$

where $T(1)$ is the time required to perform the work with one processor and $T(p)$ is the time required to perform the same amount of work with p processors.

Scaleup may be expressed as a ratio of the capacity with p processors to the capacity with one processor. This ratio is sometimes known as the *scaling factor*. It has also been called the relative capacity, $C(p)$ [Gunther 2000]. If we use the maximum throughput as a measure of the capacity, we can express the scaling factor or relative capacity as:

$$C(p) = \frac{X_{max}(p)}{X_{max}(1)}$$

where $X_{max}(1)$ is the maximum capacity with one processor and $X_{max}(p)$ is the maximum capacity with p processors. Since the maximum throughput of a system is equal to the inverse of the demand at the bottleneck resource [Jain 1990], we have:

$$C(p) = \frac{X_{max}(p)}{X_{max}(1)} = \frac{D_b(1)}{D_b(p)}$$

The demand at a resource is the total time t_b required to execute the workload at that resource (i.e., visits x service time). Thus, if we approximate the behavior of

the system by a single queue/server representing the bottleneck device, we can write:

$$C(p) = \frac{X_{max}(p)}{X_{max}(1)} = \frac{D_b(1)}{D_b(p)} = \frac{t_b(1)}{t_b(p)} \cong S(p)$$

While this approximation should be verified (see the case study below), it is a good one in most cases.

We will use $C(p)$ as a measure of the scalability of the system. Once the function (model) describing $C(p)$ has been determined, the throughput of the system with p processors can be obtained by multiplying $X_{max}(1)$ by $C(p)$.

Categories of Scalability

We will categorize the scalability of a system based on the behavior of $C(p)$. This classification scheme is similar to that proposed by Alba for speedup in parallel evolutionary algorithms [Alba 2002]. The categories are:

- *Linear*—the relative capacity is equal to the number of processors, p, i.e., $C(p) = p$.
- *Sub-linear*—the relative capacity with p processors is less than p, i.e., $C(p) < p$.
- *Super-linear*—the relative capacity with p processors is greater than p, i.e., $C(p) > p$.

Linear Scalability

Linear scalability can occur if the degree of parallelism in the application is such that it can make full use of the additional resources provided by scaling. For example, if the application is a data acquisition system that receives data from multiple sources, processes it and prepares it for additional downstream processing, it may be possible to run multiple streams in parallel to increase capacity. In order for this application to scale linearly, the streams must not interfere with each other (for example via contention for database or other shared resources) or require a shared state. Either of these conditions will reduce the scalability below linear.

Sub-Linear Scalability

Sub-linear scalability occurs when the system is unable to make full use of the additional resources. This may be due to properties of the application software, for example if delays waiting for a software resource such as a database lock prevent the software from making use of additional processors. It may also be due to properties of the execution environment that reduce the processing power of additional processors, for example overhead for scheduling, contention among processors for shared resources such as the system bus or communication among processors to maintain a global state. These factors cause the relative capacity to increase more slowly than linearly.

Super-Linear Scalability

At first glance, super-linear scalability would seem to be impossible—a violation of the laws of thermodynamics. After all, isn't it impossible to get more out of a machine than you put in? If so, how can we more than double the throughput of a computer system by doubling the number of processors?

The fact is, however, that super-linear scalability is a real phenomenon. The easiest way to see how this comes about is to recognize that, when we add a processor to a system, we are sometimes adding more than just a CPU. We often also add additional memory, disks, network interconnects, and so on. This is especially true when expanding clusters. Thus, we are adding more than just processing power and this is why we may realize more than linear scaleup. For example, if we also add memory when we add a processor, it may be possible to cache data in main memory and eliminate database queries to retrieve it. This will reduce the demand on the processors, resulting in a scaleup that is more than can be accounted for by the additional processing power alone.

The next section discusses several models that exhibit these categories of behavior.

Models of Scalability

This section discusses four models of scalability:

1. Linear scalability
2. Amdahl's law
3. Super-Serial Model
4. Gustafson's Law

The Super-Serial Model is an extension of Amdahl's Law and has appeared in the context of on-line transaction processing (OLTP) systems [Gunther 2000], in beowulf-style clusters of computers [Brown 2003] and others. The other three were developed in the context of speedup for parallel algorithms and architectures. Amdahl's Law and the Super-Serial Model, describe sub-linear scalability. Gustafson's Law, describes super-linear scalability. These models were developed in a different context, but as we illustrate they apply to Web applications.

Other models of scalability, such as memory-bounded speedup [Sun and Ni 1993], are available but are beyond the scope of this paper.

Linear Scalability

With linear scalability the relative capacity, $C(p)$, is equal to the number of processors, p.

$$C_L(p) = p \qquad 1$$

For a system that scales linearly, a graph of $C(p)$ versus p is a straight line with a slope of one and a y-intercept of zero [$C(0)=0$]. Figure 2 shows a graph of Equation 1.

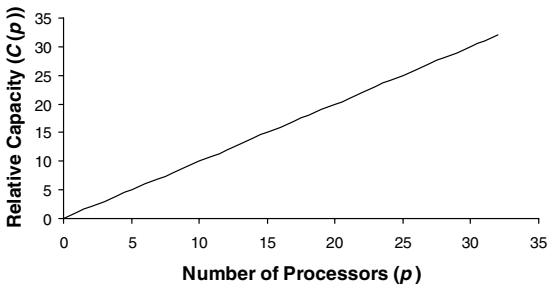

Figure 2: Linear Scalability

Note that our definition of $C(p)$ does not allow for linear scalability with a slope that is not equal to one, i.e. $C(p) = kp$ where $k \neq 1$. If this were possible, $C(1)$ could be different from one. But, since

$$C(1) = \frac{X_{max}(1)}{X_{max}(1)} = 1$$

k must be equal to one.

This means that both sub-linear and super-linear scalability must, in fact, be described by *non-linear* functions. While measurements at small numbers of processors may appear to be linear, measurements at higher numbers of processors will reveal the non-linearity.

This also means that linear extrapolations of "near-linear" results, such as that in our opening example, can be misleading. Since the actual function is necessarily non-linear, these extrapolations will overestimate the scalability of the system if the slope of the extrapolated line is less than one and underestimate it if the slope is greater than one.

Amdahl's Law

Amdahl's Law [Amdahl 1967] states that the maximum speedup obtainable from an infinite number of processors is $1/\sigma$ where σ is the fraction of the work that must be performed sequentially. If p is the number of processors, t_s is the time spent by a sequential processor on the sequential parts of the program, and t_p is the time spent by a sequential processor on the parts of the program that can be executed in parallel, we can write the Speedup for Amdahl's Law, S_A:

$$Speedup = S_A = \frac{T(1)}{T(p)} = \frac{t_s + t_p}{t_s + t_p/p}$$

$$S_A = \frac{1}{\frac{t_s}{t_s + t_p} + \frac{t_p}{t_s + t_p}\left(\frac{1}{p}\right)} = \frac{p}{\sigma + \pi/p}$$

Here, σ is the fraction of the time spent on the sequential parts of the program and π is the fraction of time spent on the parts of the program that can be executed in parallel. Since $\pi = 1 - \sigma$:

$$S_A = \frac{p}{1 + \sigma(p-1)}$$

Or, using the equivalence between speedup and scaleup,

$$C_A(p) = \frac{p}{1 + \sigma(p-1)} \qquad 2$$

If $\sigma = 0$ (i.e., no portion of the workload is executed sequentially), Amdahl's Law predicts unlimited linear scaleup (i.e., $C_A(p) = p$). For non-zero values of σ, the scaleup will be less than linear. Figure 3 shows a comparison of linear scalability with Amdahl's Law scalability for $\sigma = 0.02$.

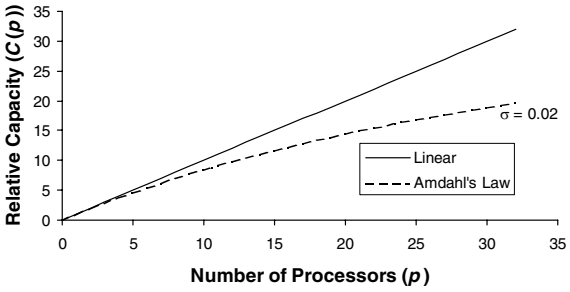

Figure 3: Amdahl's Law versus Linear Scalability

The maximum speedup that can be obtained, even using an infinite number of processors is:

$$\lim_{p \to \infty} S_A = \frac{1}{\sigma}$$

This means that, if the serial fraction of the workload is 0.02, the maximum speedup that can be achieved is 50—and it will take an infinite number of processors to achieve that! Amdahl's argument was that, given this limitation, a fast single-processor machine is more cost-effective than a multiprocessor machine.

As Figure 3 shows, there are diminishing returns for adding more processors. The penalty increases as σ

increases. For σ=0.20, Amdahl's Law predicts that adding a second processor will yield a relative capacity of 1.67. That is, the maximum throughput with two processors will be 1.67 times that with one processor. If, instead of adding a second processor, we replace the single processor with one twice as fast, the throughput will then be exactly twice that with the slower processor. This is because the faster processor reduces the time required for both the serial and parallel portions of the workload. Because of this, it is generally more cost-effective to use a faster single processor than to add processors to achieve increased throughput in cases where Amdahl's Law applies.

Super-Serial Model

Gunther [Gunther 2000] points out that Amdahl's Law may be optimistic in cases where there is interprocessor communication, for example to maintain cache consistency among processors. In these cases if, when an update is needed, a processor sequentially sends updates to each of the $p - 1$ other processors and the time required to process and send a message is t_c, we have the super-serial capacity for p processors, $C_S(p)$:

$$C_S(p) = \frac{T(1)}{T(p)} = \frac{t_s + t_p}{t_s + t_p/p + t_c(p-1)}$$

Or, after some algebra:

$$C_S(p) = \frac{p}{1 + \sigma[(p-1) + \gamma p(p-1)]} \qquad 3$$

where γ is the fraction of the serial work that is used for interprocessor communication. This result is identical to the Amdahl's Law result with an extra term in the denominator for overhead due to interprocessor communication. Gunther has called Equation 3 the Super-Serial Model [Gunther 2000].

The term in Equation 3 that contains γ grows as the square of the number of processors. This means that, even if the overhead for interprocessor communication is small, as the number of processors increases, the communication overhead will eventually cause $C(p)$ to reach a maximum and then decrease. Figure 4 shows a comparison of Equation 3 for σ = 0.02 and various values of γ with linear scalability and Amdahl's Law.

Gustafson's Law

For certain applications, it was found that speedups greater than that predicted by Amdahl's Law are possible. For example, some scientific applications were found to undergo a speedup of more than 1,000 on a 1,024 processor hypercube. Gustafson [Gustafson 1988] noted that Amdahl's Law assumes that the parallel fraction of the application ($\pi = 1 - \sigma$) is constant, i.e., independent of the number of processors. Yet, in many

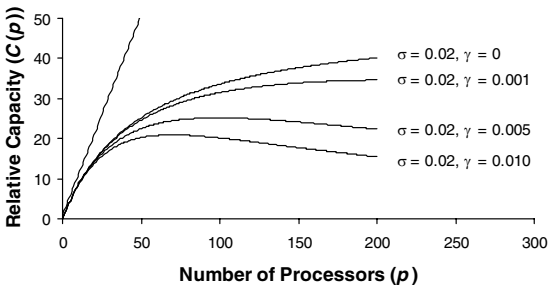

Figure 4: Super-Serial Model

cases, the amount of parallel work increases in response to the presence of additional computational resources but the amount of serial work remains constant. For example, with more computing power, matrix manipulations can be performed on larger matrices in the same amount of time. In these cases, π (and, therefore, σ) is actually a function of the number of processors.

If t_s and t_p are the times required to execute the serial and parallel portions of the workload on a parallel system with p processors, a sequential processor would require a time of t_s + (t_p x p) to perform the same work. Gustafson termed this *scaled speedup* which is described in the following equations:

$$\textit{Scaled Speedup} = S_G = \frac{t_s + (t_p \times p)}{t_s + t_p}$$

$$S_G = p + \sigma'(1 - p) \qquad 4$$

where σ' is the serial fraction of the work performed on p processors. Equation 4 is known as Gustafson's Law or the Gustafson-Barsis Law.

Gustafson's Law describes *fixed-time* speedup while Amdahl's Law describes *fixed-size* speedup.

As with Amdahl's Law, Gustafson's Law also applies to scalability. We can not use the formulation in Equation 4 directly, however. Equation 4 describes the speedup as the ratio of the time required to execute the workload on a system with p processors to that required to execute the *same* amount of work on a single processor. This is not a ratio that is likely to be measured, however. We are more likely to have measurements of the maximum throughput at various numbers of processors. Thus, to use Gustafson's Law for web application scalability, we need to express it in terms of $C(p)$, the ratio of the maximum throughput with p processors to the maximum throughput with one processor.

The demand with one processor is $t_s(1) + t_p(1)$ and the maximum throughput is therefore:

$$X_{max}(1) = \frac{1}{t_s(1) + t_p(1)}$$

Gustafson's Law assumes that the parallel portion of the workload increases as the number of processors. Thus, the total demand with p processors is $t_s(1) + (t_p(1) \times p)$. However, this demand is spread over p processors, so the average demand per processor is:

$$D_b(p) = \frac{t_s(1) + (t_p(1) \times p)}{p}$$

Note that the average demand per processor is a decreasing function of p. This is because only one processor can execute the serial portion of the workload. If the degree of parallelism is such that the application is able to make use of this additional capacity, each processor beyond one will be able to execute more parallel work than tp(1), resulting in super-linear scaling. This can occur in Web applications when loading a page results in multiple concurrent requests from the client to retrieve information, such as gifs and other page elements. Additional processors enable those requests to execute in parallel.

Under these conditions, the average maximum throughput per processor is:

$$\frac{X_{max}(p)}{p} = \frac{1}{D_b(p)} = \frac{p}{t_s(1) + (t_p(1) \times p)}$$

and the maximum throughput for p processors is:

$$X_{max}(p) = \frac{p^2}{t_s(1) + (t_p(1) \times p)}$$

Using these results, we can write the relative capacity for Gustafson's Law as:

$$C_G(p) = \frac{p^2}{p + \sigma'(1)(1 - p)} \qquad 5$$

where $\sigma'(1)$ is the serial fraction of the work with one processor.

As the value of $\sigma'(1)$ approaches zero, this function approaches linear scalability. In fact, for small values of $\sigma'(1)$, Equation 5 is difficult to distinguish from linear scalability. With non-zero values of $\sigma'(1)$, the second term in the denominator is negative for values of p greater than one, however. This means that $C(p)$ will increase faster than p giving super-linear scalability.

Figure 5 shows a graph of $C(p)$ versus p for Gustafson's Law with two values of $\sigma'(1)$. As the figure shows, at small values of $\sigma'(1)$ (e.g., 0.01) Gustafson's Law is difficult to distinguish from linear scalability. At higher values of $\sigma'(1)$, however, the curve definitely shows its non-linearity.

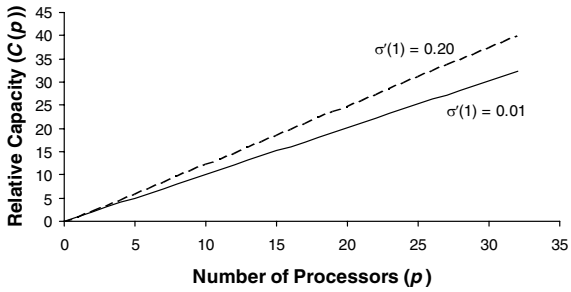

Figure 5: Gustafson's Law

The next section illustrates the applicability of these models with a case study.

CASE STUDY

This case study is based on the Avitek Medical Records (MedRec) sample application developed by BEA Systems, Inc. [BEA 2003a]. This application is an educational tool that is intended to illustrate best practices for designing and implementing J2EE applications. It has also been used to demonstrate the scalability of BEA's WebLogic Server[1] [BEA 2003b].

MedRec is a three-tier application that consists of the following layers [BEA 2003a]:

- *Presentation layer*—The presentation layer is responsible for all user interaction with MedRec. It accepts user input for forwarding to the application layer and displays application data to the user.
- *Application layer*—The application layer encapsulates MedRec's business logic. It receives user requests from the presentation layer or from external clients via Web Services and may interact with the database layer in response to those requests.
- *Database layer*—The database layer stores and retrieves patient data.

Users interact with MedRec via a browser using HTTP requests and responses. External clients may also use MedRec as a Web Service. The presentation and application layers reside on one or more application serv-

1. WebLogic Server is a trademark of BEA Systems, Inc.

ers. The database layer resides on a separate database server. Figure 6 shows a schematic view of the MedRec application.

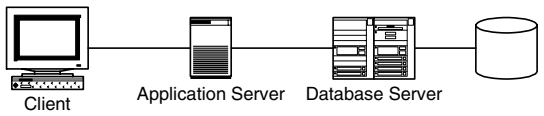

Figure 6: MedRec Application Configuration

This case study is based on measurements of this application published by BEA Systems, Inc. [BEA 2003b]. These measurements were made to demonstrate the scalability of BEA's WebLogic Server. Because of this, no specific performance requirements were specified.

Table 2 shows the demand for each of the resources in the system. The application server CPU has the highest demand and is therefore the bottleneck resource.

Table 2: Resource Demand

Resource	Demand
Application Server CPU	0.0227
Database Server CPU	0.000168
Disk	0.00138

We begin by examining the validity of the single-queue approximation for describing the behavior of the system. The following sections then explore vertical and horizontal scaling characteristics of this application.

Single Queue Approximation

This analysis is based on measured data for an application server with a single 750 MHz processor. The database server was an 8-processor (750 MHz) machine with RAID disks. The measurements appear in Table 3. Note that the high number of transactions per second for a single client is due to using a think time of zero for these measurements. We recommend a more realistic think time for your measurement studies. There are a few other aspects of the measurements that we question, but we are unable to confirm their validity because we did not conduct these measurement studies.

From the maximum throughput and the throughput with one client, we can calculate the demand at the bottleneck resource, D_b and the total demand, D_T [Jain 1990].

$$D_b = \frac{1}{X_{max}} = \frac{1}{43.96} = 0.0227 \; sec$$

Table 3: Measured Throughput versus Number of Clients

Number of Clients	Transactions per Second
1	41.39
4	43.96
10	42.78
20	43.54
40	42.74
80	43.01
100	43.05
Max TPS	43.96
Appserver CPU Utilization at Max TPS	98%
DB Server CPU Utilization at Max TPS	<1%
DB Server Disk Utilization at Max TPS	6%

$$D_T = \frac{1}{X(1)} = \frac{1}{41.39} = 0.0242 \; sec$$

Note that the total demand with one user is the sum of the demands in Table 2, 0.0242 sec. Also note that the bottleneck demand is 93.8% of the total demand. With such a large percent of the overall demand attributable to the bottleneck resource, the single-queue approximation is a good one.

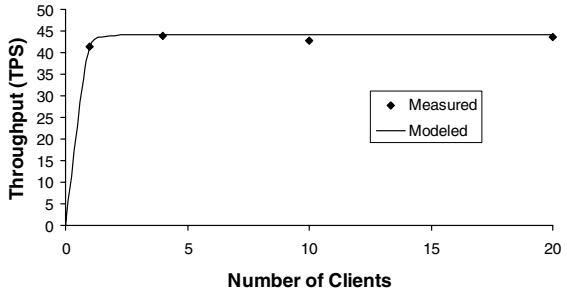

Figure 7: Measured versus Modeled Data

Using the results for D_b and D_T, we can construct a simple system model using a single queue/server that represents the bottleneck resource—in this case, the application server CPU. Figure 7 shows a portion of the measured data for the MedRec application on the single-processor server along with the modeled curve for a QNM solution using a single queue/server representing the bottleneck resource. The model was constructed and solved using *SPE•ED*[1]. As the figure indicates, the single queue approximation is a good one for this system (because the Application Server CPU dominates the demand).

Vertical Scalability

To demonstrate the vertical scaling characteristics of this application, throughput versus number of simulated clients was measured for 1-, 2-, 4-, and 8 processor application server configurations. The application server is a 750 MHz platform capable of holding up to 24 processors. Utilizations for the application and database server CPUs as well as the database server disk were also measured. Table 4 shows the results of these measurements.

Table 4: Measured Throughput for Vertical Scaling

Number of Processors	1	2	4	8
Max TPS	43.96	78.74	142.51	216.35
Number of Clients at Max TPS	4	40	20	40
Appserver CPU Utilization at Max TPS	98%	97%	95%	91%
DB Server CPU Utilization at Max TPS	<1%	1.30%	2.25%	3.91%
DB Server Disk Utilization at Max TPS	6%	12%	19%	28.24%

Note that the number of clients at Max TPS decreases at 4 processors. This is one of the questionable measurements that we found. You can see in Table 3 that the maximum transactions per second fluctuates between 4 and 80 clients, but remains close to 43 transactions per second.

Amdahl's Law Regression analysis determines that Amdahl's Law provides a good fit to this data with a σ of 0.0881 (r^2 = 0.992). This value of s indicates that 8.8% of the workload must be performed sequentially.

The maximum number of processors that can be installed in this server is 24. Thus the maximum value of $C(p)$ that can be obtained is:

$$C(24) = \frac{p}{1+\sigma(p-1)} = \frac{24}{1+0.0881(24-1)} = 7.93$$

The maximum throughput with one processor, $X_{max}(1)$, is 43.96 transactions per second so the maximum throughput with 24 processors would be:

$$X_{max}(24) = C(24) \times 43.96 = 348\ tps$$

The limit on $C(p)$ for Amdahl's Law is $1/\sigma$. Thus, even with an infinite number of processors, the maximum value of $C(p)$ that could be obtained is 11.4 for a maximum throughput of 500 transactions per second.

Super-Serial Model Regression analysis determines that the Super-Serial Model also provides a good fit to this data. This is not surprising since the Super-Serial Model is an extension of Amdahl's Law with an extra term for interprocessor communication.

The values of the super-serial parameters obtained from a regression analysis are: σ = 0.0787 and γ = 0.0164 (r^2 = 0.993). This indicates that, according to the Super-Serial Model, 7.9% of the work must be performed sequentially. Approximately 1.6% of that sequential work is used for interprocessor communication.

The value of $C(p)$ predicted by the Super-Serial Model at 24 processors is:

$$C(24) = \frac{24}{1+0.0787[23+0.0164\times 24\times 23]} = 6.82$$

which gives a maximum throughput of 299 transactions per second.

Overall Evaluation Figure 8 summarizes the measured data as well as the maximum throughput versus number of processors predicted by both Amdahl's Law and the Super-Serial Model.

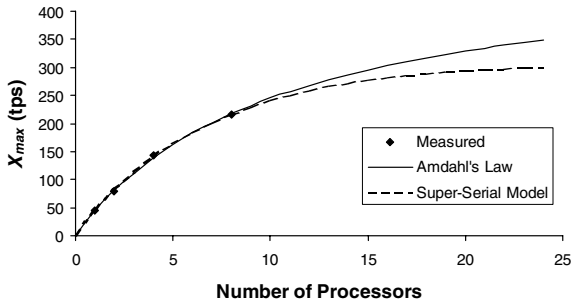

Figure 8: Summary of Modeled and Measured Data

As Figure 8 shows, there is little difference between Amdahl's Law and the Super-Serial Model in the region covered by the measured data and both models provide a reasonable fit. The values of *r*-squared from the regression analysis indicate a slightly better fit for the Super-Serial Model (r^2 = 0.993) than Amdahl's Law (r^2 = 0.992).The difference is small, however, and the increase in r^2 may be due to the extra degree of freedom introduced by the additional parameter in the Super-Serial Model rather than an improved fit.

1. *SPE•ED* is a trademark of Performance Engineering Services Division, L&S Computer Technology, Inc.

Without additional information, it is not possible to say which model fits the data better. Measurements at higher numbers of processors would help resolve the ambiguity. Knowledge of the software architecture could also help select the most appropriate model. For example, if we know that there is a shared state that must be maintained among the processors, the Super-Serial model would be the most likely choice.

In view of this, we consider the Amdahl's Law result of 348 transactions per second to be upper limit on the capacity that can be obtained by scaling this system vertically and the Super-Serial result of 299 transactions per second to be a lower bound.

Horizontal Scalability

In this section, we explore the horizontal scaling characteristics of the MedRec application. In this study, each node contains four 400 MHz processors. Rather than adding more processors to a node, additional nodes are added to scale the system.

Measurements of throughput versus number of simulated clients were made for 1-, 2-, 3-, and 4-node configurations. The database and network configurations were the same as those used for the vertical scaling study. Table 5 shows the results of these measurements.

Table 5: Measured Throughput for Horizontal Scaling

Number of Nodes	1	2	3	4
Max TPS	100.49	208.66	313.06	418.76
Number of Clients at Max TPS	40	40	40	100
Appserver CPU Utilization at Max TPS	95.09%	95.58%	95.35%	95.22%
DB Server CPU Utilization at Max TPS	1.70%	3.64%	6.21%	9.70%
DB Server Disk Utilization at Max TPS	13.94%	27.79%	40.58%	53.25%

Regression analysis shows that both Gustafson's Law and the linear model provide good fits to the measured data. Regression analysis based on Amdahl's Law and the Super-Serial model do not yield good fits to the experimental data. In addition, both analyses give negative values for the serial parameters. Thus, Amdahl's Law and the Super-Serial model are not appropriate for this data.

Gustafson's Law Regression analysis indicates that Gustafson's Law provides an excellent fit to the horizontal scaling measurements with $\sigma'(1)=0.0477$ (r^2 = 0.9999). This indicates that approximately 4.8% of the work with one processor is performed sequentially. Since this amount of work is constant, as more processors are added, this fraction decreases.

Linear Scalability The linear model also provided an excellent fit to the horizontal scaling measurements (r^2 = 0.9997). The slope of the regression line is 1.039.

Table 6 shows the measured values of maximum throughput along with the values predicted by Gustafson's Law and the linear model.

Table 6: Measured Versus Modeled Throughput

Number of Nodes	Measured	Gustafson's Law	Linear
1	100.49	100.49	100.49
2	208.66	205.89	200.98
3	313.06	311.36	301.47
4	418.76	416.86	401.96

Note that, as discussed earlier, for linear scalability the slope must be one. Thus, a slope of one was used to calculate the linear predictions in Table 6. Figure 9 shows a graphical comparison of the measured and modeled throughput for up to ten processors.

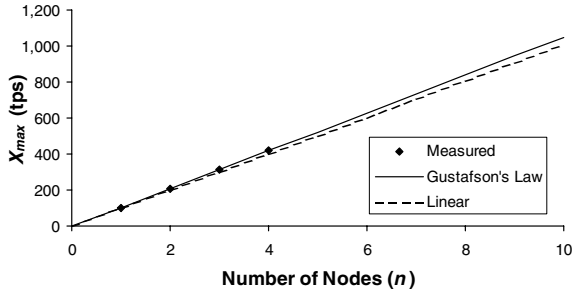

Figure 9: Measured versus Modeled Data

As Table 6 and Figure 9 indicate, the Gustafson's Law estimates more nearly match the measured values. However, an additional measurement at a higher number of processors would help distinguish between these two models.

Secondary Bottlenecks Both models predict that we can expect considerably more throughput from the horizontal scaling strategy than from the vertical. However, it is still important to be careful when extrapolating. For example, if our performance requirement is 1,000 transactions per second, Gustafson's Law predicts that using ten nodes will give a maximum throughput of 1,050 transactions per second and the linear model

predicts a maximum throughput of 1,005 transactions per second.

These projections assume that no other bottleneck resource will limit our ability to achieve the required throughput with ten nodes, however. Looking at Table 5, we see that removing the application server CPU bottleneck leaves the database server disk as the resource with the highest utilization. The demand at this resource is 0.00132 seconds which agrees with the value of 0.00138 obtained in the vertical scalability measurements to within experimental error. This corresponds to a maximum throughput of 757 transactions per second. Therefore, to achieve the performance requirement of 1,000 transactions per second, it will be necessary to either upgrade to faster disks or add a second disk to the database server.

Of course, adding only ten nodes will mean that we are operating at near one-hundred percent utilization on the application server CPU. This will result in unacceptably long response times, so additional nodes should be added to reduce the CPU utilization to a more reasonable number.

Case Study Discussion

The results of the analysis indicate a significant difference in the behavior of the system for vertical versus horizontal scaling. Since the application software is the same in both cases, the difference must be due to the platforms. The most likely explanation for this difference is the presence of the MP Effect when scaling vertically. The MP Effect is a loss of computing capacity that occurs when adding processors to a single platform. This loss of capacity is due to additional overhead and/or contention between processors for shared system resources (e.g., the system bus) [Artis 1991], [Gunther 1996]. Gunther has shown that Amdahl's Law and the Super-Serial Model apply to SMP scaling with super-serial effects becoming more significant as the amount of interprocessor communication increases [Gunther 1993].

It is tempting to generalize this result and conclude that horizontal scaling is superior to vertical scaling in all cases. However, the scalability of a system depends on the characteristics of *both* the application and the execution environment. In this case, it appears that the software was structured into units that could execute independently in parallel. If a larger percent of the workload were serial, both the vertical and horizontal scalability would have followed Amdahl's Law or the Super-Serial model. In addition, the good horizontal scalability indicates that back-end database interactions were effectively parallelized so that there was little or no serial contention there either. This will not be true of all applications. In particular, for distributed applications that must maintain a common state (e.g., databases) there will be overhead to maintain coherence. In these cases, interprocessor communication will be a significant factor. For such applications, the enhanced communication efficiency of a bus versus a network may favor vertical scaling on a single platform. Each application/platform combination should be evaluated individually.

This case study also emphasizes the importance of thoroughly understanding the system before committing to a scaling strategy. Secondary bottlenecks (such as the database disk here) are a fact of life. It is important to know what they are, how they impact the scalability of the system, and how costly they are to remove before undertaking expensive system upgrades.

Finally, it should be noted that the data published by BEA Systems, Inc. provides a good example of the type of information that is useful for understanding and evaluating scalability.

Economics of Scalability

Scalability is an economic as well as a technical issue. In many cases there are alternative scaling strategies that will meet performance requirements. The choice among them should then be based on cost. The importance of cost in meeting performance requirements is implicit in the inclusion of cost figures in many benchmark reports (see, e.g., [TPC]).

These costs are rarely simple, one-time costs. Adding more hardware means that there will be costs for purchase or lease, software licenses, maintenance contracts, additional system administration, facilities, and so on. The timing of these expenditures is likely to be highly dependent on the scaling strategy. For example, one strategy may require frequent, small upgrades while another may require fewer, more expensive ones. In order to make an unbiased comparison of the alternatives, it may be necessary to convert the costs to current dollars. Techniques for this are discussed in many places, including [Reifer 2002] and [Williams and Smith 2003].

It is important to consider all costs associated with the scaling strategy. In some cases, hardware expenditures may be dwarfed by costs for support software and middleware [Mohr 2003].

Costs of upgrading are also subject to discontinuities as the amount of hardware is increased. For example, at some point adding a server may require hiring an additional system administrator or expanding the facility to accommodate the additional footprint.

Don't forget that scalability isn't just a hardware issue. It is often easy to increase scalability with a different (initial) software architecture [Williams and Smith 2003]. In this case study, a design alternative that reduces the CPU consumption of the application could also improve scalability. The modification would produce a new version of the application with new scalability characteristics.

Note that the most cost-effective choice for meeting a given performance requirement might not be the one with the highest overall scalability.

Summary and Conclusions

Scalability is one of the most important quality attributes of today's distributed software systems. Yet, despite its importance, scalability in these applications is poorly understood. This paper has presented a model-based view of scalability in Web and other distributed applications that is aimed at removing some of the myth and mystery surrounding this important software quality.

We use the relative capacity or scalability factor

$$C(p) = \frac{X_{max}(p)}{X_{max}(1)}$$

as a measure of the scalability of a system. Scalability is classified according to the behavior of $C(p)$ as:

- *Linear*—the relative capacity is equal to the number of processors, p, i.e., $C(p) = p$.
- *Sub-linear*—the relative capacity is less than p, i.e., $C(p) < p$.
- *Super-linear*—the relative capacity is greater than p, i.e., $C(p) > p$.

A key consequence of using $C(p)$ as the metric for scalability is that for linear scalability, the slope of the line must be equal to one. This means that both sub-linear and super-linear scalability must, in fact, be described by *non-linear* functions. While measurements at small numbers of processors may appear to be linear, measurements at higher numbers of processors will reveal the non-linearity.

This also means that linear extrapolations of "near-linear" results, such as that in our opening example, can be misleading. Since the actual function is necessarily non-linear, these extrapolations will overestimate the scalability of the system if the slope of the extrapolated line is less than one and underestimate it if the slope is greater than one.

This paper has reviewed four models of scalability that are applicable to Web and other distributed applications. These are summarized in Table 7.

We have also demonstrated the applicability of these models to Web applications via a case study of a simple Web application. Analysis of measured data for the case study system indicates that its vertical scalability is best described by either Amdahl's Law or the Super-Serial Model while its horizontal scalability is best described by either Gustafson's Law or linear scalability.

Table 7: Scalability Models

Model	$C(p)$
Linear	$C_L(p) = p$
Amdahl's Law	$C_A(p) = \frac{p}{1 + \sigma(p-1)}$
Super-Serial Model	$C_S(p) = \frac{p}{1 + \sigma[(p-1) + \gamma p(p-1)]}$
Gustafson's Law	$C_G(p) = \frac{p^2}{p + \sigma'(1)(1-p)}$

The case study also demonstrates that scalability is a *system* property. In this case, the same application exhibits different scalability properties when scaling vertically or horizontally.

As this paper demonstrates, these models are applicable to Web and other distributed systems. However, due to the complexity of such systems, it is likely that some systems will not conform to any of these models. It is therefore important to determine, via analysis of measured data, that a given system follows a known model before making decisions based on predicted scalability.

Scalability is also affected by software resource constraints such as a One Lane Bridge Performance Antipattern [Smith and Williams 2002]. This paper addressed only hardware bottlenecks. This paper also considered only a single dominant workload. These techniques can be adapted to cover these other aspects of the problem.

References

[Alba 2002] E. Alba, "Parallel Evolutionary Algorithms Can Achieve Super-Linear Performance," *Information Processing Letters*, vol. 82, pp. 7-13, 2002.

[Amdahl 1967] G. M. Amdahl, “Validity of the Single-Processor Approach To Achieving Large Scale Computing Capabilities,” *Proceedings of AFIPS*, Atlantic City, NJ, AFIPS Press, April, 1967, pp. 483-485.

[Artis 1991] H. P. Artis, “Quantifying MultiProcessor Overheads,” *Proceedings of CMG '91*, December, 1991, pp. 363 - 365.

[BEA 2003a] BEA Systems, Inc., “Avitek Medical Records 1.0 Architecture Guide,” http://edocs.bea.com/wls/docs81/medrec_arch/index.html.

[BEA 2003b] BEA Systems, Inc. “BEA WebLogic Server: Capacity Planning,” http://e-docs.bea.com/wls/docs81/capplan/.

[Brown 2003] R. G. Brown, “Engineering a Beowulf-style Compute Cluster,” Duke University, 2003, www.phy.duke.edu/resources/computing/brahma/beowulf_book/.

[Gunther 2000] N. J. Gunther, *The Practical Performance Analyst*, iUniverse.com, 2000.

[Gunther 1996] N. J. Gunther, “Understanding the MP Effect: Multiprocessing in Pictures,” *Proceedings of CMG '96*, December, 1996.

[Gunther 1993] N. J. Gunther, “A Simple Capacity Model for Massively Parallel Transaction Systems,” *Proceedings of CMG ‘93*, San Diego, December, 1993, pp. 1035-1044.

[Gustafson 1988] J. L. Gustafson, “Reevaluating Amdahl's Law,” *Communications of the ACM*, vol. 31, no. 5, pp. 532-533, 1988.

[Jain 1990] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, New York, NY, John Wiley, 1990.

[Mohr 2003] J. Mohr, “SPE on IRS Business Systems Modernization,” Panel: “The Economics of SPE,” CMG, Dallas, 2003.

[Reifer 2002] D. J. Reifer, *Making the Software Business Case: Improvement by the Numbers*, Boston, Addison-Wesley, 2002.

[Smith and Williams 2002] C. U. Smith and L. G. Williams, *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*, Boston, MA, Addison-Wesley, 2002.

[Sun and Ni 1993] X.-H. H. Sun and L. M. Ni, “Scalable Problems and Memory-Bounded Speedup,” *Journal of Parallel and Distributed Computing*, vol. 19, no. 1, pp. 27-37, 1993.

[TPC] Transaction Processing Council, www.tpc.org.

[Williams and Smith 2003] L. G. Williams and C. U. Smith, “Making the Business Case for Software Performance Engineering,” *Proceedings of CMG*, Dallas, December, 2003.

[Williams and Smith 2002] L. G. Williams and C. U. Smith, “PASA^{SM}: An Architectural Approach to Fixing Software Problems,” *Proc. CMG*, Reno, December, 2002